

제출일: 2017.11.4

프리데이 사용일수: 3

시스템 프로그래밍 1차 과제

- Log Structured File System Profiling -

14조

컴퓨터학과 2015410089 이태현(조장)

컴퓨터학과 2015410079 이준경(조원)

차례

1. 서론

1) 배경지식

- (1) VFS
- (2) Block layer
- (3) FFS
- (4) NILFS

2) 개발환경 및 역할 분담

- (1) 개발환경
- (2) 역할 분담

2. 본론

1) 커널 수정 및 코드 작성

- (1) 작성 코드 개요
- (2) 작성한 구조체(struct bnotime) 설명
- (3) block/blk-core.c 수정내용
- (4) fs/nilfs2/segbuf.c 수정내용
- (5) 작성한 Loadable Kernel Module(lkm.c) 설명

2) Nilfs2 쓰기연산 테스트 수행

3) 결과 그래프 분석

3. 결론

4. 참고자료

1. 서론

1) 배경지식

(1) VFS(Virtual File System, 가상 파일 시스템)

VFS는 응용프로그램 및 사용자가 다양한 파일 시스템에 대해 동일한 방법으로 접근이 가능하도록 설계된 실제 파일 시스템 위의 추상 계층이다.

사용자 프로세스는 VFS를 통해 파일 시스템이나 물리 매체의 종류와 관계없이 open, read, write 등과 같은 시스템 콜을 사용할 수 있게 된다.

VFS에는 다음과 같은 네 가지의 주요한 객체들이 있다. (in-memory 객체, 커널 내 include/linux/fs.h에 정의되어 있음)

- superblock: 마운트 된 파일 시스템의 정보를 저장.
- inode: 특정 파일에 대한 정보를 저장, inode number를 가짐.
- file: 열린 파일과 프로세스 사이의 상호작용과 관련한 정보 저장, file pointer를 가짐.
- dentry: 디렉토리 항목과 대응하는 파일간 연결에 대한 정보 저장.

(2)Block Layer

어플리케이션과 파일 시스템의 사이에 위치하여 디바이스에 접근하는 추상화(abstraction)된 interface를 구현한 것을 Block Layer라고 한다. linux-4.4/block에 소스코드가 위치해 있다.

- struct * block device : 일정 크기(block) 단위로 데이터에 접근하는 장치를 block device라고 하며 일반적으로 하드 디스크와 같은 대용량 저장 장치를 뜻한다. 파일 시스템을 통해 간접적으로 접근하게 된다.

- struct * bio : sector(H/W적 관점에서 device에 접근하는 최소 단위(512B)), block(S/W적 관점에서 한 번에 접근하게 되는 단위, 디스크 상에서 연속된 sector로 이루어져있으며 주로 4KB), segment(메모리에서 device와의 I/O연산을 위한 데이터를 저장하는 영역)을 비롯해 READ, WRITE, FLUSH, FUA, DISCARD 등의 연산 및 해당하는 정보를 담고 있는 구조체.
- submit_bio() : I/O 연산의 종류(READ, WRITE) 및 해당 연산에 대한 모든 정보를 포함하는 bio 구조체를 인자로 받아 I/O operation을 수행한다.

(3) FFS(Fast File System)

①UFS(Unix File System)과 FFS

FFS는 기존의 Unix File System의 단점인 작은 블록사이즈(512Byte, throughput이 작다), data block들의 임의적 배치에 의한 seek time의 증가(inode와 data block사이의 넓은 간격)를 해결하기 위해 fragmentation기법을 적용한 큰 블록 사이즈 할당과, Cylinder 그룹의 정의 등의 방법을 통해 전체적인 성능을 끌어올린 파일 시스템이다.

②구조

- **Boot block:** 메모리에 올라가 OS를 부트하거나 초기화하는 코드를 저장한다.
- **Superblock:** 파일 시스템의 상태(크기, 개수, 미사용공간 등) 정보를 가지고 있다.
- **Inode list:** 각 파일의 위치 정보와 메타 데이터를 가지는 inode들의 모임이다.
- **Data block:** 파일의 데이터가 저장된다.
- **Cylinder group:** 각 cylinder group마다 superblock, inode, files가 배치되며, 같은 디렉토리에 속하는 파일과 inode를 같은 cylinder group에 배치함으로써 디스크 arm의 이동거리를 줄여 seek time을 낮추게 된다.

(4) NILFS2(New Implementation of a Log-structured File System2)

①NILFS2의 정의 및 특징

NILFS2는 리눅스 커널에 적용되는 log-structured file system(LFS)이다. log-structured file system은 log(circular buffer)에 데이터와 메타 데이터를 순차적으로 저장하는 파일 시스템으로, 기존 파일 시스템들이 데이터의 spatial locality를 최대화하고 최적의 배치를 이루기 위해 많은 노력을 기울여 상대적으로 많은 seek time이 소모되는 데 반해 LFS는 단지 log의 head에 계속해서 순차적으로 데이터를 쌓아 나감으로써 seek time을 최소화 해 쓰기 속도를 크게 향상 시키는 기법이다. LFS는 또한 다음과 같은 특징들을 지니고 있다.

- wear-leveling: LFS는 주로 플래시 메모리에서 쓰이게 되는데, 파일을 쓰는 과정자체가 wear-leveling이 가능하게끔 동작하게 되는 이점이 있기 때문이다. (큐를 돌면서 쓰기가 실행되기 때문에 각각의 블록에 쓰기 연산이 균일하게 할당됨)
- snapshot: LFS는 연속적인 로그에 덮어쓰기 없이 모든 데이터를 append하는 방식으로 write가 수행되는 데 이것은 copy-on-write 기법(데이터 수정 발생 시 원본 데이터의 사본을 만들어 따로 저장하는 방식)으로 snapshot을 가능하게 한다. LFS는 이를 통해 crash로 인한 data loss를 최소화 할 수 있으며, 사용자가 실수로 덮어쓰거나 지운 파일을 복구하는 것 또한 가능하게 해준다.

②NILFS2의 구조 및 동작 원리

- inode map: LFS에서는 inode number를 inode의 현재 위치와 매핑 시켜주는 global inode map을 사용하여 data copy시 발생하는 overhead를 줄여준다.

- segment와 checkpoint: LFS에서 디스크는 많은 수의 데이터 블록과 inode들, 그리고 가장 최근의 inode map을 가지고 있는 몇 개의 커다란 segment로 나뉘어져 있는데, 파일시스템의 시작부분에 위치한 superblock은 가장 최근에 쓰여진 segment를 가리키고 있다. 현재 쓰이고 있는 segment가 꼭 차거나 주기적으로 segment의 데이터를 disk에 쓰게 되는 동작이 있으면 superblock의 head segment number는 다음 segment를 가리키도록 업데이트 되고 이것을 checkpoint라고 부른다. 이 checkpoint가 설정되면, checkpoint 이전에 일어난 모든 일은 filesystem에 반영되게 된다.
- compaction(garbage collection): LFS는 old version을 포함한 모든 데이터를 저장하기 때문에 저장소가 쉽게 가득 차게 될 수 있다. 따라서 주기적으로 log의 끝부분에서부터 사용되지 않는 segment들을 정리하는 compaction 과정이 실행된다. segment를 정리하기 위해 현재의 inode table을 통해 각각의 block이 최신의 version인지 확인한다. 최신의 version이라면 log의 head로 block 및 inode, inode table을 옮기고 그 segment를 재사용하게 된다. segment table은 각각의 block이 어디에 속하는지에 대한 정보를 담고 있다.

2) 개발환경 및 역할분담

(1) 개발환경

가상머신 : Oracle VM VirtualBox

운영체제 : Ubuntu 16.04.1 LTS (64bit)

커널 : linux-4.4.0

파일시스템 : ext4, nilfs2

사용언어 : C

하드웨어 스펙 :

이태현 : CPU : i5-6600 RAM : 16GB(Virtual Box: 2 Core, 4GB RAM)

이준경 : CPU : i5-3317U RAM : 8GB(Virtual Box: 1 Core, 2GB RAM)

(2) 역할분담

이태현: Circular Queue 구현 및 lkm 작성, 파일시스템 이름 출력

이준경: 파일시스템 이름 출력, iozone 테스트 수행, 보고서 작성

2. 본론

1) 커널 수정 및 코드작성

(1) 작성 코드 개요

(1) -1. block number 출력

- submit_bio함수를 호출 할 때, struct bio를 사용한다.
- bio 내부 struct bvec_iter b_iter 변수 에 있는 bi_sector 변수가 512바이트 섹터의 device address라는 것을 확인할 수 있었다.

(1) -2. time 출력

- struct timeval, struct tm구조체를 사용한다.
- 시간을 불러올 때, do_gettimeofday 함수를 호출하여 timeval 구조체를 초기화 한다.
- Time_to_tm 함수를 호출하여 tm 구조체로 만들어줄 수 있다.

(1) -3. NILFS segbuf.c superblock 관련 변경

- NILFS는 자신만의 자료구조를 사용한다.

- Nilfs_segbuf_submit_bio에서 submit_bio함수를 호출하여 작업을 수행한다.
이 때, NILFS의 자료구조가 bio 형태로 반환된다.
- Struct nilfs_segment_buffer 의 sb_super가 super_block을 저장하고 있는
것을 확인할 수 있었다.

(1) -4. 파일시스템 이름 출력

- struct bio bio struct
block_device bi_bdev -> struct super_block bd_super -> struct
file_system_type s_type -> name(찾고자 하는 값)
- VFS에서는 객체지향 방식 프로그래밍을 통해서 사용자는 각 하드웨어의
동작을 알 필요 없이 동작할 수 있게 설계되어있다.
- 각 블록오퍼레이션은 block_device에 대한 정보를 가지고 있다.
- block_device는 super_block에 대한 정보를 가지고 있다.
- Super block을 통해서 각 파일시스템에 맞는 VFS의 명령을 수행하게 된다.
- Super_block 내부에 file_system_type 에 대한 정보가 저장되어 있다.
- file_system_type 구조체 내에 파일 시스템의 이름이 저장되어 있다.

(2) 작성한 구조체(struct bnotime) 설명

구조체는 include/linux/struct.h에 생성하여 block/blk-core.c와 추후 만들
lkm에서 자료구조를 공유할 수 있게 생성하였다.

```
struct bnotime{
    unsigned long long block_number; //block number를 저장
    struct tm *time; //tm 구조체를 사용하여 time 저장
    struct bio **bio; //bio 구조체를 사용하여 bio 저장
    int hour; //시간의 시 저장
    int min; //시간의 분 저장
    int sec; //시간의 초 저장
    struct block_device *bdev; //get_super를 통해서 super block을 받아올 때
    사용, 처음부터 super block을 저장하지 않는 이유는 blk-core.c에서 설명
```



```
};
```

(3) block/blk-core.c 수정내용

(3) -1. 새로 만든 함수 설명

```
#define CQ_BUF_SIZE 16384 //Circular Queue의 크기를 16384으로 지정
int CQ_BUF_HEAD = 0;
int CQ_BUF_TAIL = -1;
int CQ_length = 0;
struct bnotime CQ[CQ_BUF_SIZE]; //Circular Queue구현

EXPORT_SYMBOL(CQ_BUF_HEAD);
EXPORT_SYMBOL(CQ_BUF_TAIL);
EXPORT_SYMBOL(CQ_length);
EXPORT_SYMBOL(CQ);
void CQ_enqueue(unsigned long long bn, struct tm *t, struct bio **blockio)
{
    if(CQ_BUF_HEAD == (CQ_BUF_TAIL + 1) % CQ_BUF_SIZE && CQ_length >
0)
    {
        CQ_BUF_HEAD = (CQ_BUF_HEAD + 1) % CQ_BUF_SIZE;
        CQ_length--;
    }
    CQ_BUF_TAIL = (CQ_BUF_TAIL + 1) % CQ_BUF_SIZE;
    CQ[CQ_BUF_TAIL].block_number = bn;
    CQ[CQ_BUF_TAIL].hour = t->tm_hour;
    CQ[CQ_BUF_TAIL].min = t->tm_min;
    CQ[CQ_BUF_TAIL].sec = t->tm_sec;
    CQ[CQ_BUF_TAIL].bio = blockio;
    CQ[CQ_BUF_TAIL].bdev = NULL;
    CQ[CQ_BUF_TAIL].bdev = (*blockio)->bi_bdev;
    CQ_length++;
}
```

CQ_BUF_HEAD, CQ_BUF_TAIL, CQ_length, CQ는 각 변수로 lkm에서

Queue에 대한 정보를 가져올 수 있도록 EXPORT_SYMBOL 을 이용하여서

다른 파일에서도 이 변수에 대해 접근할 수 있도록 만들었다.

또한 deque함수가 필요 없도록, modulo(%)연산을 이용하여서 큐가 꽉 차는

순간부터 HEAD 또한 하나씩 증가시켜 큐가 순환할 수 있도록 구현하였다.

(3) -2. Submit_bio 함수 수정

```
if (rw & WRITE) {
    struct timeval time;
    struct tm broken;
    do_gettimeofday(&time);
    time_to_tm(time.tv_sec, 0, &broken);
    CQ_enqueue((unsigned long long)bio->bi_iter.bi_sector, &broken, &bio);
    count_vm_events(PGPGOUT, count);
}
```

위에서 생성한 CQ_enqueue 함수를 이용하기 위해서 쓰기 순간이 발생하는 순간,
즉 submit_bio 함수 내에 rw & WRITE 속성이 true 일 경우에 CQ_enqueue를
호출할 수 있도록 하였다. 여기서 timeval, tm 구조체와 do_gettimeofday를
이용하여서 현재의 시간을 저장할 수 있도록 하였다.

(4) fs/nlfs2/segbuf.c 수정내용

```
bio->bi_bdev->bd_super = segbuf->sb_super;
```

위 코드를 nlfs_segbuf_submit_bio 함수 내에서 submit_bio 함수가 호출되기
전에 넣어 줌으로써 각 블록이 super block에 대한 정보를 가질 수 있도록
수정하였다.

(5) 작성한 Loadable Kernel Module(lkm.c) 설명

```
extern int CQ_BUF_HEAD;
extern int CQ_BUF_TAIL;
extern struct bnotime CQ[CQ_BUF_SIZE];
extern int CQ_length;
static void lkm_print_buffer(void)
{
    printk(KERN_INFO "lkm_print_buffer\n");
    int index = 0;
    int i = 0;
    for(i = CQ_BUF_HEAD; i != CQ_BUF_TAIL; i = (i + 1) % CQ_BUF_SIZE)
    {
        index = i;
        struct bnotime temp = CQ[index];
```

```

        if (strcmp(get_super(temp.bdev)->s_type->name, "nilfs2")
== 0) {
            printk(KERN_INFO "fs: %5s\t",
get_super(temp.bdev)->s_type->name);
            printk(KERN_INFO
"b_no : %10llu %2d:%2d:%2d\n",
                    temp.block_number,
                    temp.hour,
                    temp.min,
                    temp.sec);
        }
    }
}

```

lkm에서는 lkm_print_buffer 함수를 통해서 CO에 있는 내용들을 출력한다.

큐를 출력하는 for문을 생성하고, CO에 저장되어있는 bdev 를 통해서 super_block을 가져온다. 이 때, super_block은 바로 데이터를 가져올 수 없고, get_super 함수를 사용하여야 한다. Super_block을 가져오면, s_type->name에 접근하여 파일 시스템의 이름을 출력하게 된다. Enqueue시 저장해주었던 block number를 출력해주고, 각 구조체에 존재하는 hour, min, sec의 정보를 출력하게 된다. 이 때, 컴퓨터의 local time과 출력되는 time이 다른데, 이는 처음 do_gettimeofday는 UTC를 이용하여 시간을 출력하기 때문에 시차에 따른 보정을 하지 않고 출력하기 때문이다.

Block/blk-core.c에서 EXPORT_SYMBOL를 사용하였던 변수들을 사용하기 위해 변수들을 extern으로 선언하였다.

2) 쓰기 연산 테스트 수행방법 및 결과

- i. 수정된 커널소스를 컴파일한 후 부팅한다.

```

Sudo make -1 #
Sudo make install
reboot

```

- ii. 모듈을 올린다.

```

Insmod lkm.ko

```

- iii. 가상 디스크 파일을 만들고 nilfs로 포맷한다.

```
dd if=/dev/zero of=./diskfile bs=1024 count=1000000
```

```
mkfs.nilfs2 ./diskfile
```

(1GB 할당)

- iv. Disk를 I/O 디바이스로 등록한다.

```
losetup /dev/loop0 ./diskfile
```

- v. 디렉토리를 만들어 디스크를 마운트한다.

```
mkdir nilfs_disk
```

```
mount -t nilfs2 /dev/loop0 nilfs_disk/
```

- vi. Izone 테스트를 수행한다.

```
sudo ./iozone -Ra -g 1G -i 0 -f /home/junkyoun/nilfs_disk/test
```

(nilfs_disk에 대해 결과를 엑셀 형식으로 출력하는 write/rewrite only 자동

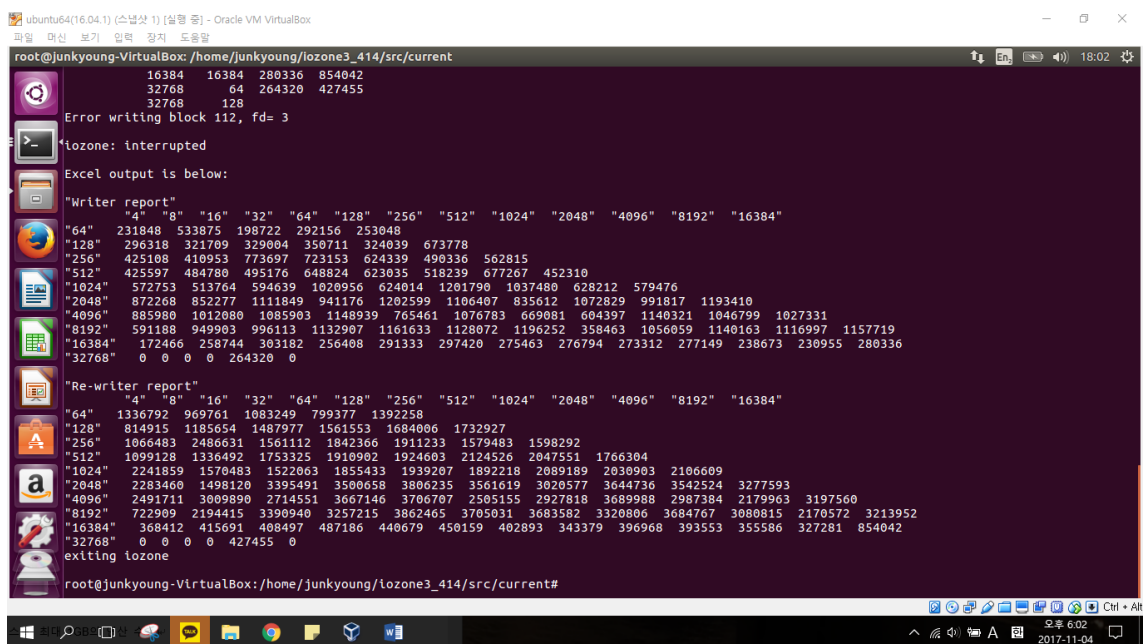
테스트, 최대 1GB의 연산 수행)

- vii. Proc 파일시스템을 통해 연산 결과를 읽어온다.

```
Cat /proc/block_print/blockprint
```

- viii. 커널 로그를 확인한다.

```
Cat /var/log/kern.log
```



```
root@junkyoun-VirtualBox: /home/junkyoun/iozone3_414/src/current
16384 16384 280336 854042
32768 64 264320 427455
32768 128
Error writing block 112, fd= 3
iozone: interrupted
Excel output is below:
"Writer report"
"4" "8" "16" "32" "64" "128" "256" "512" "1024" "2048" "4096" "8192" "16384"
"64" 231848 533875 198722 292156 253848
"128" 296318 321709 329804 358711 324039 673778
"256" 425108 410953 773697 723153 624339 490336 562815
"512" 425597 484780 495176 648824 623035 518239 677267 452310
"1024" 572753 513764 594639 1020956 624014 1201790 1037480 628212 579476
"2048" 872268 852277 1111849 941176 1202599 1106407 835612 1072829 991817 1193410
"4096" 885980 1012080 1085903 1148939 765461 1076783 669081 604397 1140321 1046799 1027331
"8192" 591188 949903 996113 1132907 1161633 1128072 1196252 358463 1056059 1140163 1116997 1157719
"16384" 172466 258744 303182 256408 291333 297420 275463 276794 273312 277149 238673 230955 280336
"32768" 0 0 0 264320 0
"Re-writer report"
"4" "8" "16" "32" "64" "128" "256" "512" "1024" "2048" "4096" "8192" "16384"
"64" 1336792 969761 1083249 799377 1392258
"128" 814915 1185654 1487977 1561553 1684006 1732927
"256" 1066483 2486631 1561112 1842366 1911233 1579483 1598292
"512" 1099128 1336492 1753325 1910902 1924603 2124526 2047551 1766304
"1024" 2241859 1578483 1522063 1855433 1939207 1892218 2089189 2039903 2106609
"2048" 2283460 1498120 3395491 3500658 3806235 3561619 3020577 3644736 3542524 3277593
"4096" 2491711 3089890 2714551 3667146 3706707 2505155 2927818 3689988 2987384 2179963 3197560
"8192" 722909 2194415 3390940 3257215 3862465 3705031 3683582 3320806 3684767 3080815 2170572 3213952
"16384" 368412 415691 408497 487186 440679 450159 402893 343379 396968 393553 355586 327281 854042
"32768" 0 0 0 427455 0
exiting iozone
root@junkyoun-VirtualBox: /home/junkyoun/iozone3_414/src/current#
```

<iozone 수행 결과>

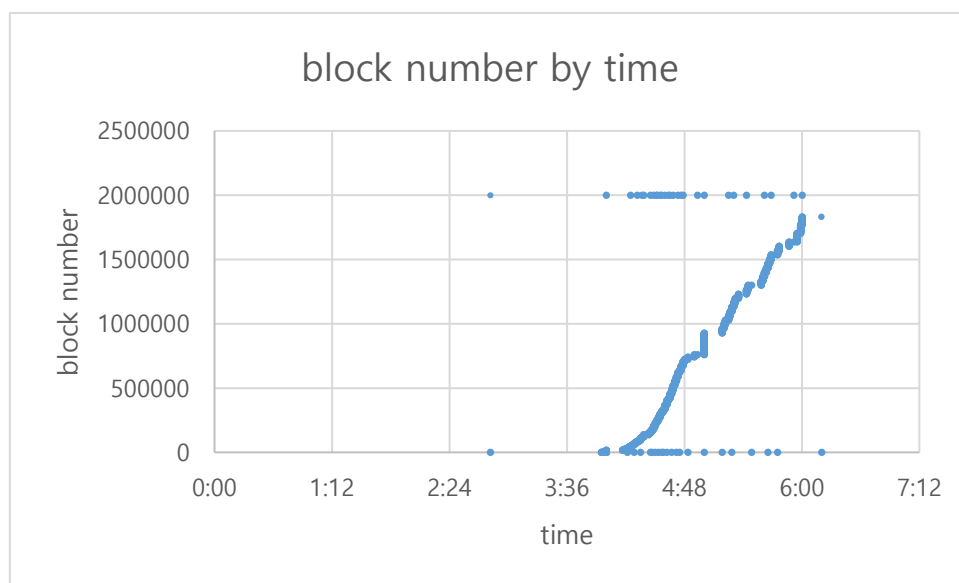
```

root@junkyoun-VirtualBox: /
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679746] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679747] b_no : 0 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679749] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679750] b_no : 1617184 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679751] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679752] b_no : 1619232 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679754] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679755] b_no : 1621280 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679756] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679757] b_no : 1622016 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679758] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679759] b_no : 1624064 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679761] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679761] b_no : 1626112 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679763] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679764] b_no : 1628160 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679765] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679766] b_no : 1630208 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679767] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679768] b_no : 1632256 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679770] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679771] b_no : 1634304 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679772] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679773] b_no : 1636352 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679774] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679775] b_no : 1638400 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679777] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679777] b_no : 1640448 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679779] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679780] b_no : 1642496 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679781] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679782] b_no : 1644544 9: 0:14
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679783] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679784] b_no : 1646240 9: 0:19
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679786] fs: nilfs2
Nov 4 18:02:19 junkyoung-VirtualBox kernel: [ 643.679787] b_no : 1646608 9: 0:24
root@junkyoun-VirtualBox: /#

```

<kern.log>

3) 결과 그래프의 해석 및 FFS와의 차이점



<block number by time>

Nilfs2의 쓰기 연산 수행 시 시간에 따라 block number가 연속적으로 단방향 증가하는 것을 볼 수 있다. 이는 nilfs의 쓰기연산이 log structured file system의 구조에 따라 계속해서 로그의 앞부분에 데이터 및 inode를 append 시키는 과정임을 알 수 있게 해준다. Rewrite의 경우에도 copy-on-write의 기법대로 앞부분에 사본을 만들어 계속 append 하는 것으로 생각할 수 있다.

반면에 FFS는 쓰기 연산 시 파일, 디렉토리 및 inode 위치가 일정한 규칙에 따라 정해져 있기 때문에 블록 넘버가 순차적으로 증가하지 않고 여기저기 흩어져서 나타날 것으로 예상된다. 뿐만 아니라 비록 같은 디렉토리 내의 파일은 같은 실린더에 기록하여 파일의 쓰기 위치를 정하기 위한 seek time이 기존 UFS보다는 빠르겠지만 seek time이 거의 없다고 할 수 있는 nilfs2에 비해서는 같은 양의 쓰기 연산 시 더많은 시간이 소요될 것으로 예상된다.

3. 결론

1) 과제 수행 시 어려웠던 부분 및 해결 방법

(1) 파일 시스템 이름 출력과정

super_block 의 struct file_system_type *s_type내에 name 변수가 과제에서 요구하는 file system name(ex. Ext4, nilfs2 등) 을 저장하고 있음을 찾아낼 수 있었다.

i . blk-core.c 에서 super_block을 struct에 바로 저장하는 경우

struct bnode(새로 만든 구조체) 에 struct super_block *sbb 변수를 만들어 CQ_enqueue부분에서 바로 sbb에 bio의 super_block을 저장하는 경우와, get_super를 통해서 저장하는 시도를 하였었다.

전자의 경우와 후자의 경우 둘 다 문제가 생겼었다. Super_block의 data에 접근을 할 수 없거나, super_block자체가 초기화 되지 않는 듯 하였다.

ii. lkm에서 *bio를 통해서 super_block에 접근하는 경우

위에서의 문제점으로 인해서 bio를 struct bnode에 저장하고, 이를 통해서 superblock에 대한 정보를 바로바로 저장하고자 하였고, 커널 컴파일 후 커널을 올리는 과정에서 부팅이 되지 않는 문제점이 발생하였었다.

iii. 해결

해결책은 bdev 변수를 생성하고 각 블록의 block_device를 저장하는 것이었다. 또한 lkm내부에서는 bdev변수를 통해서 get_super로 super_block 을 불러와 문제를 해결할 수 있었다.

5. 참고자료

A block layer introduction part 1: the bio layer (October 25, 2017, Neil Brown)

<https://lwn.net/Articles/736534/>

Snapshot101:CopyonwritevsRedirectonwrite (April 1, 2016, wcurtispreston)

<https://storageswiss.com/2016/04/01/snapshot-101-copy-on-write-vs-redirect-on-write/>

Log-structured file system

<http://www.cs.cornell.edu/courses/cs4410/2015su/lectures/lec20-lfs.html>